Technological Feasibility Analysis

WearWare Study Manager

Sponsored by Dr. Kyle Winfree and Dr. Eck Doerry
Mentored by Han Peng

HeartStep

Members:
Anton Freeman, Halcyon de la Rosa, Karina Anaya,
Keil Hubbard, Noah Olono

Table of Contents

1. Introduction

Cardiovascular diseases are a major problem and one of the leading causes of death in America. According to the CDC, about 1 in 4 deaths each year are caused by heart disease. Researchers hoping to help those already affected and prevent further deaths need to be able to examine both the cause of cardiovascular diseases and possible ways to mitigate their impact. However, the equipment used for these studies is expensive, sometimes difficult to use, and often not available for long-term use, which can lead to unreliable or biased data. Heart monitors and other sensory equipment necessary to gather physiological data for studies are often expensive and sometimes uncomfortable for the people wearing them, especially when it comes to heart monitors that use adhesives that must be worn at all times, leading to a risk of user error. This can potentially deter researchers from conducting studies or participants from joining studies, meaning that studies will lose out on important data that could help deal with this problem of cardiovascular disease. Additionally due to the short-term nature of much of this monitoring, some researchers including our clients believe that there is the potential for flawed readings as users may subconsciously alter their fitness behavior for the first few weeks when they know that they are being monitored.

Our clients Dr. Kyle Winfree and Dr. Eck Doerry plan to solve these problems by introducing an alternative to this current technology. Instead of requiring expensive and uncomfortable technology to collect heart data, they plan to allow researchers to collect this data using Fitbits. These small wrist-mounted devices are much less pricey than heart monitors, fit comfortably on the wrist to avoid the discomfort that comes with wearing a heart monitor, and could allow users to avoid short-term bias by collecting long-term data well after any beginning behavior changes have settled. However, the system they currently use to conduct this research has some significant problems. Currently their Fitbit data collection system's workflow requires them to manually sign up participants to studies one at a time, and exporting the collected data can take a very long time while at the same time preventing any other use of the system. Additionally, the strain on the system caused by these export times causes the current system to timeout and abort with minimal error reporting and no chance to resume with current progress.

To fix these issues, we propose abandoning the current system and moving to a custom solution built by us with essential functionality from the current system tailored specifically to resolve these issues and some additional needed functionality for our client and any future users. For this end we plan to create a web app that will entirely replace their current program for collecting and managing Fitbit data, and an API that will allow programmatic access for more technically skilled users to retrieve data.

This web app will contain the following features:

- Creation of "studies" so researchers can encapsulate and collect data over a given time period.
- Participant signup that can be done manually by researchers, through an email invitation, or through a link that researchers can share.
- Storage of Fitbit data from participants into a database for later use
- Continuous data exporting to prevent long, inefficient export times.
- Notification of export completion.
- Logs of previous exports and download links.
- Data access from both the web app and via an API.
- Secure login and account services for both researchers and participants.
- Data validation and error correction to ensure that there are no issues like missing data from participant mistakes

As a separate service, our web app's API will contain the following features:

- Authentication services to prevent access to unauthorized data sets.
- Easy retrieval of study and participant data in a specific JSON format.
- Effective and readable error reporting.
- Recovery options for failed operations and exports.

In this Technological Feasibility Analysis document, we will begin by outlining the major technological challenges we expect to face while building this project. The following subsections contain our analysis of each of these technical challenges including what important characteristics we want our solutions to each challenge to have, the different solutions we have found, an analysis and comparison between the solutions, and our rationale for choosing a particular solution. Finally, once we have listed our solutions we will then discuss how we plan to fit all of these individual parts together in order to form a full product.

2. Technical Challenges

We have established what we plan to build and what our web app and API will do, and now we are ready to discuss in more detail how we will build this project. At this early stage of the project, we are currently outlining the details of our current technological challenges and potential solutions to said challenges, then selecting the solution that we believe will best fit our needs.

Our current anticipated technical challenges are the following:

- A database to store collected Fitbit data.
- A frontend for users to interact with the web app.
- A backend to help the web app run.
- Hosting for our web app so people can connect.
- An API for programs built with languages like Python, Matlab, and R to connect and get data for researchers to use.
- Secure user authentication so both researchers and participants can log in and access data safely.

3. Technology Analysis

Now that we have outlined our current challenges, we need to find solutions to deal with those that will best fit our needs. In this section, we will go over the options that we have found for how to fix each section as well as what option we chose and the reasons we had for choosing them.

3.1. Database

Introduction

Figuring out the best database for our project is one of our most important decisions, as data is the backbone of the project. We need a reliable database that can handle a lot of data. Our final application should be secure and well supported when it comes to dealing with large amounts of data, as our client has indicated that we will possibly be working with data amounts in the hundreds of thousands and we must keep data secure to protect our user's information. We will be looking at several different database tools and figuring out which one would work best for our current plans. We know that our client's current system uses PostgreSQL so we will be exploring that option while also looking for a better approach.

Desired Database Characteristics

There are a few characteristics that we are looking for specifically in a database:

- Security: The database must be secure to protect its user's information. It is important that the database is safe so that users are willing to provide data to studies that collect said data with our system.
- Support: The database needs to have a good amount of support, including online resources to help its users. Support is important for when a problem occurs with the database, the more resources and help available, the easier it will be to solve the problem.
- Performance: The database needs to perform well in terms of both speed and reliability. Having good performance is important for researchers as they conduct studies since lots of data will be imported and exported out of the database and having slow speeds can impact the flow of the research.
- Documentation: Our database should be well-documented so we can fully understand it. Documentation includes reviews, forums, and other sources that can help at understanding of the database.

- Ease of Use: Ease of use is also an important characteristic for a database since if we overcomplicate things then we will have problems. Being able to understand how a database works is important to a successful workflow.

Alternatives

MySQL:

MySQL is a widely used relational database management system. Being based entirely on SQL means databases built in MySQL can be easily expanded or transferred to another RDBMS. MySQL is currently open-source and easy to use. However, as the baseline rights are owned by Oracle, an Enterprise Edition also exists, locking certain functionalities and expansions behind a paywall. MySQL currently supports 9 distinct storage engines, allowing developers to use the best one that fits with their application and underlying operating system.

MariaDB:

Forked from MySQL by the original developers after the acquisition by Oracle, MariaDB is free open-source software, meaning all functionality is available from source and always free. It can handle both small and large data tasks, and is used by major companies such as Google and Wikipedia. MariaDB was originally developed as a drop-in replacement for MySQL, but has since evolved into a true competitor. MariaDB supports 12 distinct storage engines compared to MySQL's 9. Further unique performance tools have also been implemented, such as parallel queries, allowing multiple queries at once increasing speed especially on large datasets. Additional extensions such as JSON, WITH, and KILL statements, allow for a more custom integration with a final application.

PostgreSQL:

PostgreSQL is an open-source object-relational database that significantly extends SQL. These additional features help developers build applications that handle non-standard, or poorly defined data. Additional data types can therefore be used in PostgreSQL including primitives, structs, documents, and geometry. With additional data type support it must also include significant custom SQL commands and functions for sorting, storing and retrieving this data. Although extremely powerful, these added features also make PostgreSQL one of the largest and most difficult to learn systems.

Analysis

The three databases we are considering have a number of similarities; for example, all of them are open-source and take in the same information for the most part. We want to analyze each of the three databases based on the following: Security, Support, Performance, Documentation, and Ease of Use.

Security:

Database security relies heavily on the security of the system hosting it, however choosing a DBMS that provides additional protections against unauthorized or unwanted access and modifications will protect both the developer and client. Security whitepapers are regularly provided by the development teams behind these programs and records of data breaches can be found online.

MySQL & MariaDB support similar secondary authorization tools, including password login, two-factor authentication and unique certificates. PostgreSQL includes similar login features with multiple additional encrypted access features, giving it only a slight security advantage over MariaDB and MySQL.

Documentation:

Documentation provided by each of the development groups is easily available and hosted on their individual sites. Additional documentation and notes are also frequently available along with the source code, allowing users to remain up to date.

MySQL is extremely popular and well documented as it has a significantly larger user base and documentation is maintained by Oracle. However, being tightly controlled means updates to MySQL documentation may be slower than updates to source. PostgreSQL is not new in the database industry and so has extensive documentation, however the added features mean there is more information to sort through at a given time. MariaDB is well documented on an individual functionality level, however resources have not been as well compiled or organized, meaning the information can still be found, but with additional effort.

Support:

In terms of support, MySQL is very strong, as the most popular of these alternatives, community resources are abundant, however official support is locked behind the Enterprise Edition as owned by Oracle. MariaDB, being based on MySQL can take advantage of many of those same resources, however further support for custom features is limited by the smaller user base, there is no single organization to provide definitive official support. PostgreSQL, although open-source, is considered an enterprise tool for experienced developers, there are fewer resources online than MySQL or MariaDB. Paid Enterprise support is available for PostgreSQL on a subscription basis only.

Performance:

Once again we used claims by the developers, user reviews and third-party benchmarks to evaluate performance. We must also be aware which option will make most efficient use of limited hardware. MariaDB outperforms MySQL in terms of speed for views, handling, and replication. MariaDB's parallel queries can take advantage of some slower processors compared to the other options, and scales exponentially with increased RAM and CPU speed. PostgreSQL is best at handling very large datasets, however the additional tools and larger install size may negatively impact the rest of the system

Ease of Use:

MySQL is highly documented, highly popular and the most closely based upon basic SQL, making it an ideal choice for developing simple applications quickly. MariaDB, being based on MySQL, is similarly easy to use, with most differences being in background performance; what few additional tools and commands that are available are only sometimes necessary and often self explanatory. PostgreSQL is an extremely expansive tool, with features that expand beyond the behavior of a standard RDBMS, although powerful, if unnecessary, this adds bloat and confusion.

Chosen Database Approach

The database is the root of the entire project, as such finding an appropriate tool early is very important. In reviewing the three most viable RDBMS options, MySQL has great community support and is easy to use, but falls short in extensibility and customization. MariaDB performs well in terms of speed and high data volume, explicitly meeting a need for our client. PostgreSQL can handle heavy and non-standard datasets, however is focused on enterprise and resource heavy.

	Security	Support	Performance	Documentation	Ease of Use	Average Score
MySQL	★★★★☆	★★★☆☆	★★★☆☆	★★★★★	★★★★★	★★★★☆
MariaDB	★★★★☆	★★★★☆	★★★★★	★★★☆☆	★★★★☆	★★★★☆
PostgreSQL	★★★★★	★★★☆☆	★★★☆☆	★★★★☆	★★☆☆☆	★★★☆☆

Table 3.1.1: A table showing our ratings of the different databases.

Table 3.1.1 shows the results of the analysis in terms of security, support, performance, documentation, and ease of use of all three databases. In reviewing database options it was also important to take into account client experience and preference; in multiple instances our client expressed a preference for MariaDB. Which,in addition to the client's major concerns relating to performance, became the tie-breaking variable leading to the choice of MariaDB for the application database.

Proving Feasibility

MariaDB is in use for applications both larger and smaller than this application. As we have been given an estimate of the size of datasets that must be supported, we will begin by developing a simple database with a fraction of the entries. Scaling up the data volume as development continues, using "dummy" data based on a single example provided by the client.

3.2. Web App Frontend

Introduction

Having a solid frontend will be very important for this project, as we need a frontend to display the interactive parts of our web app on a web page for our users. This will be the main way for researchers and study participants to interact with the Wearware Study Manager: researchers will use it to make and manage the studies within the web app, while participants will use it to sign up for studies.

Desired Frontend Characteristics

For this project, our desired characteristics are the following:

- Documentation: To use the described framework we first have to understand how the framework actually works, which makes solid documentation very important.
- Scalability: If we want to work more on the project after reaching our current goals, scalability is important to make sure we can expand the frontend to fit our needs.
- Performance speed: Our client brought up performance as an issue with the existing technology, so we want to make sure everything in this project runs quickly and smoothly.
- Ease of Use: We want to be able to easily use our framework as we develop it without having too many difficulties or complications.

Alternatives

Angular:

Angular is a frontend framework based on Javascript that uses HTML as a template language and extends its functionality, which makes it better suited for web apps like the one we are making compared to base HTML. We discovered Angular through a teammate's past experience with it, as well as from websites online discussing various types of frontend frameworks, and found that it would work well for our purposes. Angular is an open-source framework that was developed by Google along with several other open-source contributors and has been around for about 11 years since its release in October of 2010. Angular is used for web development and has been used in projects like Gmail, Microsoft Office, and PayPal.

React:

React is a Javascript library used for handling the view controller of an application that can also be used as a frontend framework. Unlike Angular, React uses an XML-like syntax called JSX to render web pages instead of using a template language. Our team members have quite a bit of experience with React, so this was our initial idea on what to use to build this website. React is an open-source framework that is developed and maintained by Facebook along with other open-source contributors. It has been around for around 8 years, having been released in May of 2013. React has been used by major companies like Facebook and Netflix to host their websites along with other popular apps like Instagram.

Vue:

Vue is a frontend framework that also extends HTML to make it better suited for responsive web apps. We discovered Vue while looking at comparisons between Angular and React, as it seemed to be brought up quite a bit alongside them as a comparable tool for web development. Vue is an open-source framework that was initially developed by former Google developer Evan You as a more lightweight alternative to Angular, and was released about 7 years ago in February 2014. Vue is used by several major companies such as Gitlab, Nintendo, and NASA for developing their own web applications.

Analysis

To evaluate each of these options, we investigated all of the different features that these frameworks contain and evaluated how said features would affect the various desired characteristics listed above.

Documentation:

We first examined each of these framework's documentation as well as how others online evaluated them. React's documentation seems to be somewhat on the

short side. Much of the functionality in React is in separate third-party libraries, which means that the documentation is far more split up and not as consistent as the documentation of Angular and Vue. After looking at Angular and Vue's documentation, it seems that while Angular has much more documentation due primarily to the amount of features it has compared to the more lightweight Vue, Vue's documentation is much less complicated than Angular's and thus easier to understand and learn from.

Scalability:

We then looked at the options that each framework provides for scalability, including the way that its code can be structured for reuse. When it comes to scalability, Vue falls short of the other options; as it is intended for more lightweight web apps, it does not work as well as Angular or React when it comes to scaling the website up. This is mostly due to its use of templates instead of modules or components, making code difficult to reuse. Meanwhile, Angular as a framework is built to offer a great amount of scalability with its use of modules that can be easily reused to make sure the website is able to grow without having to rewrite key parts. React's component-based structure also offers a good deal of scalability, albeit somewhat less than Angular's module-based approach.

Performance:

For performance, we looked at the features that would mainly affect both runtime and startup time and compared them to each other. We also looked at an article comparing the three frontend solutions' performances. Both Vue and React use a virtual DOM that allows for faster runtime when compared to the real DOM that Angular uses. Furthermore, Vue is much more lightweight than both React and Angular, allowing it to achieve better speed and performance than the other two.

Ease of Use:

To analyze ease of use we looked at what we needed to know to use each of the different options, as well as what each of our members already knew. Angular requires knowledge of Typescript, which was not very familiar to us. Additionally, in our teammate's past experience with Angular we found it to be more difficult to use than other options. React's Javascript options seem to be much easier to use than the Typescript that Angular requires, and all of our team members planning to work on the frontend have familiarity with the React system. While none of our team members have used Vue in the past, its template-based approach and use of Javascript means that we will be able to quickly learn and transfer over our skills from React, and a quick review of its tutorial in the documentation indicated that it would be very easy to use.

Chosen Frontend Approach

In summary, choosing between each of these three options was difficult due to their individual advantages and weaknesses. Some pros of each option include Angular' excellent scalability, the team already being familiar with React, and Vue's speed, documentation, and ease of use. However, each of these options also has cons. Angular is difficult to use, React's reliance on third-party libraries means that it has somewhat spotty documentation, and Vue's lightweight nature causes it to fall behind when it comes to scalability and code reuse. Table 3.2.1 shows a list of each option ranked by the desired characteristics provided above.

	Documentation	Scalability	Performance	Ease of Use	Average Score
React	★★★☆☆	★★★★☆	★★★★☆	★★★★☆	★★★★☆
Angular	★★★★☆	★★★★★	★★★☆☆	★★☆☆☆	★★★☆☆
Vue	★★★★★	★★★☆☆	★★★★★	★★★★★	★★★★★

Table 3.2.1: A table showing our ratings of the different frontend architectures.

Overall, this was a very close decision. However, when we look at all the ratings provided in Table 3.2.1, Vue seems to beat the competition in all areas except scalability. Vue beats both Angular and React when it comes to its documentation, performance and speed, and ease of use. Because with our current requirements scalability is not our main priority, we can overlook that and choose to use Vue as our main frontend framework.

Proving Feasibility

Moving forwards, we will need to show how Vue will work for our website. To accomplish this goal, we will first create a prototype frontend in Vue for our client to test out and evaluate. This prototype will not be connected to anything else and will simply exist to show what the typical user flow would look like in our web app using this framework. If he likes it, we will then move on to the next phase and attach Vue to our backend and database to make a mostly functioning website that we can test. Our main demos that we plan to include in this attached website will be showing off study creation, study usage, and participant sign up.

3.3. Web App Backend

Introduction

As we are working to improve and extend an existing system, our web application must be supported by a strong backend solution. To address our client's concerns, our backend solution will need to support interaction with our dynamic frontend as well as provide secure read/write access to a custom relational database. This implementation must be extensible to support "stretch-goal" future requests and other long-term growth.

Desired Backend Characteristics

The characteristics that our client desires and that we need to build a quality system include the following:

- Performance: Our client desires high data speed transfer, as his old technology has issues with slow data exporting.
- Extensibility: Our backend should be able to be frequently updated to help deal with any issues pertaining to speed, data volume and security.
- Documentation: Good documentation will make sure we can build our project well, as a poorly built system will function the same or worse than the old system.
- Ease of use: Likewise, an overcomplicated system due to user misuse will experience poor performance even with a stronger base technology.

Alternatives

Django:

Django is a relatively young web application tool, built to allow writing web servers in Python. Django's main focus is to provide developers with the fastest path from concept to execution. Python is an interpreted language not originally designed for the web and requires multiple third-party tools to support asynchronous processes. That makes it the slowest running among all the given options. Django comes pre-packaged with tools for static site pages, and limited dynamic content.

Our client's current technology uses Django, which we believe contributed to our client's complaints about export speed and export volume.

NGINX:

NGINX is a highly flexible JavaScript based web server, first built to address issues with C based servers. According to its website, NGINX is currently running over 450 million sites. In addition to the free open-source software offering and extensive documentation, NGINX offers *NGINX Plus,* a subscription based service for enterprise implementation that provides additional support, features, and an uptime guarantee.

ExpressJS:

ExpressJS is a wrapper library for implementing web server applications with Node.js. As ExpressJS extends the original Node.js, it does benefit from Node.js' focus

on asynchronous/ non-blocking run-time and pre-built HTTPS support. However this also means that the developer must have at-least some prior experience with Node.js. Additional Node.js libraries or custom implementations are required to support REST API services and/or SMTP.

Apache TomCat:

Apache TomCat is an extremely popular open-source Java Servlet implementation. TomCat is popular among extremely large sites, and being Java based allows it to take advantage of the years of growth and documentation that Java has accrued. However, utilizing Java means that updates and upgrades are slower and will often require a full recompile which will either need a redundant server or introduce downtime.

Required Feature Support Breakdown:

Feature Support	NGINX	ExpressJS	Apache TomCat	Django
Host Dynamic GUI	Yes	With Modification	Yes	With Modification
Database Integration	Yes	Yes	Yes	Yes
SSL/ TLS Encryption	Yes	Yes	Yes	Yes
Authentication API Support (REST)	Yes	Yes	Yes	Yes
Integrated Email Tools (SMTP)	Yes	No	No	Yes
Microservices Support	Yes	Yes	Yes	Yes

Table 3.3.1: A breakdown showing each of our required features for each backend architecture.

Analysis

Maturity:

Maturity in our context both refers to both the chronological age and completeness of the feature set. For this project an “immature” program is one in which major feature updates and/or major runtime changes are still being made. A “mature”

program is one in which a comprehensive and complete feature set exists and updates relate to performance and security. Among our options, NGINX and Apache TomCat are the most mature, with large active user bases, histories of successful updates, and many major features for the modern web already supported that allow us to develop a solution with fewer concerns that a future update may cause our application to fail. However, Apache Tomcat's latest major update moved away from the Java servlet to Jakarta and existing systems have had to make major code changes to comply.

Extensibility:

Extensibility refers to the ability, ease and extent with which new processes or tools can be added. For this we explored not only the optional libraries provided within the base application, but also the volume of custom libraries and extensions created by the user-base. Among our options, NGINX and ExpressJS both allow for significant extension at the lowest development cost. ExpressJS also allows the implementation of nearly all Node.js functions and abilities, while NGINX has a history of wide use, meaning the community has created many resources for nearly all reasonable extensions.

Performance:

Data transfer speed, data transfer volume, and amount of possible concurrent sessions are all part of our client's concern. We compared our options using user experience, along with benchmark results from both the developers and third-party reviewers. Django rates last in this category by design, where NGINX and Apache TomCat once again stand out; with the sheer volume of sites supported, these frameworks have had to adapt to increasing size and speed requirements making them obvious leaders. NGINX due to its smaller base install size and newer roots can take better advantage of fewer hardware resources giving it a slight lead in this use case.

Documentation:

All of the given choices have extensive high-quality documentation. However, of the alternatives, the more "mature" alternatives of NGINX and TomCat as shown above have documentation that has been refined, clarified and is less subject to change. This once again allows for additional confidence that the solution we build on top of that framework can last. Apache Tomcat however, due to the latest move away from Java has historical documentation that still does not reflect the new Jakarta requirements and standards.

Ease of Use:

This is exactly as it sounds: the ease at which a developer can learn and begin working with a given framework. Django shines here as it was developed for the purpose of prototyping and/or low time-to-market, followed by NGINX which uses

baseline JavaScript conventions. ExpressJS requires additional knowledge of Node.js and Apache TomCat is an extremely large and old tool that has recently moved to an entirely new servlet language with some features requiring a steep learning curve.

Chosen Backend Approach

All of our potential alternative backend solutions are FOSS (Free Open-Source Software) with equal support for REST APIs, SSL/TLS encryption, database integration, and microservices support such as being dockerized. Therefore our decision comes down to the few features that are not immediately met and the level at which they meet the client's and our desired characteristics as shown in Table 3.3.2.

	Maturity	Extensibility	Performance	Documentation	Ease of Use	Average Score
Django	★★☆☆☆	★★★☆☆	★★☆☆☆	★★★★☆	★★★★★	★★★☆☆
NGINX	★★★★★	★★★★☆	★★★★★	★★★★☆	★★★★☆	★★★★★
ExpressJS	★★★☆☆	★★★★☆	★★★☆☆	★★★★☆	★★★★☆	★★★☆☆
Apache TomCat	★★★☆☆	★★★☆☆	★★★★☆	★★★☆☆	★★☆☆☆	★★★☆☆

Table 3.3.2: A table showing our ratings of the different backend architectures.

NGINX meets all of our requirements, as well as being highly rated for speed and extensibility, and some members of our team have already limited previous experience with NGINX. Additionally, as we will be using a JavaScript based frontend framework, we believe it is best to use a consistent programming language throughout development, making it both easier, cheaper, and faster for us or any future teams to update or upgrade this product. Therefore, we have made the decision to move forward with NGINX as our backend solution.

Proving Feasibility

NGINX has already been used for many sites that perform an extremely similar function to what we are going to build. For this particular service the most important aspect is how well it is able to integrate with our specific database and frontend. To prove NGINX is an acceptable solution, we will build extremely lightweight versions of those services to support the implementation testing on a small scale.

3.4. API

Introduction

With APIs, our main challenge is looking for the option that will best support our project's requirements for big data transfer from our database to an end user to the best of its ability. Our project focuses on easily searching and returning large amounts of data from study results back to the user.

Desired API Characteristics

With our APIs, we are looking for the following characteristics:

- Fast performance: The API should be able to transfer data quickly so that researchers can get their data as soon as possible.
- Scalability: The API needs to be flexible enough to handle anything from a small amount of data to thousands of entries from a study, as indicated by our client's estimate of studies having possibly hundreds of thousands of data points.
- Security: The API must be able to keep the information sent by it secure so that personal data from research participants is not leaked.
- Ease of use: Ease of use is important so that we do not waste too much time trying to figure out how to get these APIs to work and troubleshooting these APIs if they have issues.

Alternatives

REST:

Representational State Transfer Architectural, also known as REST, is a set of rules and styles to define how an API should work. For an API to conform to these styles, it first needs an uniform interface that ensures that the same piece of data belongs to only one uniform resource identifier. This ensures that the resources aren't too large and will contain every piece of information that the client needs. Next, REST requires client-server decoupling, which means the client and server applications are completely independent of each other. Following this requirement, REST requires that the API is stateless, meaning that it does not require any server-side sessions and the server isn't allowed to store any of the data related to the client request. REST was introduced and defined in 2000 by Roy Fielding, and is used in many public web services like Twitter, Instagram, and Spotify.

SOAP:

Simple Object Access Protocol (SOAP) is a messaging protocol for interchanging the data in the decentralized and distributed environment. SOAP follows a standardized approach that specifies how to encode XML files to return via the API. Starting with the required blocks for the XML formatting, SOAP uses an envelope structure for the starting tags of the message, with a body block that contains the XML data that the server transmits to the receiver. Optional additions to the XML file include the header that contains the attributes of the message and allows extension of the message and the fault that carries information about errors during the message transmission. SOAP was designed and released in June 1998 by Dave Winer, Don Box, Bob Atkinson and Mohsen Al-Ghosein for Microsoft and is used for enterprise and high-security apps such as PayPal and Salesforce.

Analysis

Scalability:

REST is a very flexible software architecture style that has loose guidelines, which makes it easier to scale for modern web applications thanks to the separation between the client and server. Its flexibility especially stands out when compared to SOAP, which has strict standard guidelines that leave little room for flexibility. Thus, REST is a much better API for scalability, as SOAP must keep with its strict standards for security purposes.

Performance:

REST has higher performance overall, thanks to its caching of the data that is not altered or dynamic. SOAP requires more bandwidth and computing power because of its use of a complex XML format, which makes it slower to run while going through its WS-Security and SSL protocols. Meanwhile, REST uses JSON formatting which is faster to run and only has HTTPS protocols to go through.

Ease of Use:

REST is a very popular API due to its ease of use and thanks to its flexible messaging formatting. REST accepts many formats such as JSON, HTML, XML, Python, PHP, and plain text. Meanwhile, SOAP is as popular as REST because of the fact that it is not as easy to use with its very limited XML formatting options.

Security:

SOAP is commonly used because of its support for WS-Security for transmissions along with SSL, which are security measures in the service that protects it from external attacks. WS-Security ensures that communication between the client and server is not interrupted by any unauthorized third-party, and SSL allows for secure

communication from client to server by exchanging public keys to create a secure connection. REST meanwhile has very less security based on its loose guidelines and includes only HTTPS protocols in its security, which make it easy to have firewalls and proxies without modifications.

Chosen API Approach

REST and SOAP both have their advantages and disadvantages. While REST is a very popular option and has much more flexibility and speed, SOAP's strengths lie in its security and standardization. Table 3.4.1 shows a comparison between the two based on our previously stated desired characteristics.

	Scalability	Performance	Ease of Use	Security	Average Score
REST	★★★★☆	★★★★★	★★★★★	★★★☆☆	★★★★☆
SOAP	★★☆☆☆	★★★☆☆	★★★☆☆	★★★★★	★★★☆☆

Table 3.4.1: A table showing our ratings of the different API options.

After comparing the two, it is clear that REST has many of the qualities we are looking for in an API: it is easier to use, more flexible with its loose guidelines which allow for increased scalability, and has faster performance. Even with its limited security options, it still comes out on top in all other aspects.

Proving Feasibility

As we continue further with this project, we will need to show how our REST API will actually work. After our database is filled with dummy data given to us by our client, we will begin testing the API and see what features we need to implement. We will then create a demo that shows off the API's features and how it performs when getting data collected during studies.

3.5. User Authentication

Introduction

An integral part of our solution is the ability to access and view study and participant data using our web app and API systems. Every researcher needs to be able to see their own study's data and be excluded from viewing the data from other studies they are not a part of. People entrust their personal fitness data to the study researchers using this platform, and they expect that their data will not be available to the general public or to other unauthorized parties. To ensure the safety of our user's data we need to have some way to prevent public or unauthorized access and assign or revoke access to a given party. We will do this by implementing user accounts protected using an HTTP authentication and authorization scheme which will allow us to make sure that every user can easily prove their identity and that they are allowed to access a given resource.

Desired Authentication Characteristics

When looking at this challenge we decide that our main desired characteristics for this part of the project are the following:

- Reliability: Our solution must be reliable and available at all times for users so that they can access the system whenever they need.
- Security: Any attempt to intercept credentials or gain unauthorized access should be very difficult so our user's confidential data can be kept safe away from prying eyes.
- Ease of Use: Our system implementation should be understandable and well documented for ease of use by both maintainers and end users.
- Documentation: It is very likely that any system we choose will have some vulnerabilities or issues that we will need to address later. Because of this, we must have extensive and understandable documentation so that we can find out the cause of our mistakes and build our system properly to avoid these issues.

Alternatives

HTTP Basic Authentication:

One of the oldest methods in which passwords are encoded in plain text and sent over the network is as base64 streams. This scheme is not often used or preferred because messages are easy to decode which makes it vulnerable to many types of

attacks. This standard has been around since at least 1999 and the latest version is specified in RFC 7617 written by Julian Reschke, a contributor at GreenBytes in 2015.

HTTP Digest Authentication:

HTTP Digest Authentication is similar to HTTP basic authentication, with the main difference being that it hashes the user credentials using a hashing method such as md5 before sending them to the server for verification. This is more popular because any intercepted message is hashed and so the credentials can only be forged by breaking or matching the hash. The standard is defined in RFC 7616 which overwrote the 1999 RFC 2617.

Session Based Authentication:

Session Based Authentication uses session data created when the user first logs in to the server to authenticate subsequent connections until timeout, browser close, or logout. This is achieved by sending a cookie with session data to the server with every successive auth request. While not a standalone scheme this system is often used in conjunction with some of the other alternatives listed. The main goal of this is to delay the need for users to manually re-authenticate until session timeout. This standard is defined in RFC 6896.

Web Token Based Authentication:

Web Token Based Authentication creates an encrypted token similar to a hotel key card which is sent by the server and allows temporary access until the token is changed or expired. Defined in RFC 7519, this allows for temporary access and authentication.

OAuth2 or Open Authorization:

OAuth2 allows authentication using a third-party authentication service provided by websites like Google and Facebook. Users can log in using their third-party accounts with authentication happening on that third-party server and have access to their accounts on our server after successful validation. As an open standard, the documentation and source can be found on GitHub and the oauth.net site, as well as RFC 6749 which defines the latest standard for OAuth2.

Analysis

When considering the best approach for this topic, we used the characteristics defined above to compare and contrast each solution.

Reliability:

All the systems mentioned allow for the basic reliability needed for this project. However, HTTP basic, HTTP digest, and session-based authentication are heavily reliant on us having our own authentication server up and running at all times whereas with an OAuth system we could rely on the uptimes of a third-party server. This would mean more thought would be required to choose a reliable and trusted provider.

Security:

For security purposes, we should exclude HTTP basic because transmitting login data in plain-text or base64 is no longer best practice, as it is easily intercepted and decoded. HTTP digest does offer more security because it uses hashes which allow a further layer of encoding, but it also takes on the vulnerabilities of the chosen hash function. While we might be able to solve this problem by implementing an additional acceptable encryption scheme such as AES, we would need further research and time for correct and secure implementation. Looking at the next option, a web token system would allow us to issue credentials to users. However this would require us to manage, store, and issue new credentials. OAuth2, on the other hand, is considered very secure and of the few vulnerabilities we would need to anticipate, only securing our connection to the authentication server would be of any difficulty. Additionally, since authentication is handled by the third-party we would only need to store account information.

Ease of use:

All these systems would be relatively simple to implement and use on both the server and client applications. OAuth2 is more complex in some instances because it utilizes redirects and a third-party system, but due to our teammate's past experiences with it and the ease that we would have with testing it out due to all of its login systems already being implemented on the third-party system's side, we believe that it would be easier to use and implement than the other systems.

Documentation:

OAuth2 ranks the best by far in this category as there are an abundance of up-to-date tutorials and readable documentation available because of its widespread use in the industry. As for the other options, there is a bit of documentation in the form of the above-mentioned RFC documents. However, we were unable to find much more in the way of official documentation with current practices and information.

Chosen Authorization Approach

There was a lot for us to consider when researching this topic, especially considering that some of us have not had any experience with this subject in the past at a similar level. After all our research, we have decided that a system using OAuth2 would be our best option because it best meets our required characteristics thanks to its high levels of security, reliability, ease of use, and documentation. Our ratings for each option can be found in Table 3.5.1, where based on our scores we decided that OAuth2 was the best fit for our system. The data from the table shows that OAuth2 has the highest overall average and fits our needs the best, so we have selected it as our chosen solution.

	Security	Reliability	Ease of use	Documentation	Average Score
HTTP Basic	★☆☆☆☆	★★★★★	★★★★★	★★★☆☆	★★★☆☆
HTTP Digest	★★★☆☆	★★★★★	★★★★★	★★★☆☆	★★★★☆
Web Token	★★★☆☆	★★★★★	★★★☆☆	★★★☆☆	★★★☆☆
OAuth2	★★★★☆	★★★★☆	★★★★★	★★★★★	★★★★★

Table 3.5.1: A table showing our ratings of the different authentication options.

Proving Feasibility

Our plan for testing and validating this choice is to create and integrate a simple OAuth2 service into our frontend demo using Google's simple OAuth2 framework to make sure all authentication can be handled as we envisioned in our planning phase. From there, we will look into ways that we can use the tokens from OAuth2 to make secure user accounts and also add those to our demo as well.

4. Technology Integration

Now that we have outlined the solutions to our technical challenges, we must consider the ways that our solutions will interact with each other to create a fully functioning web app. Using the following system diagram, we can examine a quick overview of how our web app will function once everything has been put together.

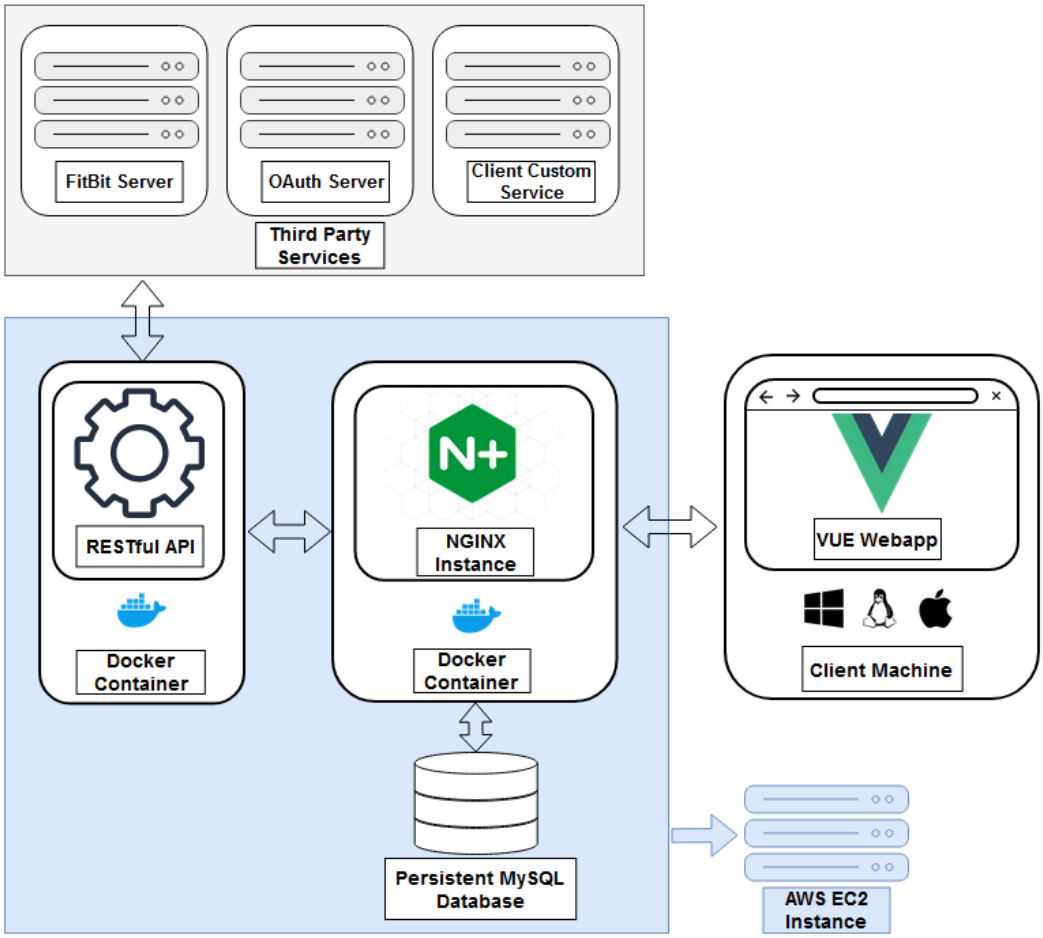

Figure 4.1: A diagram showing how all the components of our project will work together.

To begin, we will have an AWS server running Docker as specified by our client, inside of which we will contain the different parts of the web app. We will have our NGINX backend inside of the Docker container which will connect to our database in order both to gather data from it and feed in the data from our frontend and the Fitbit servers. The data gathered from the backend will both be transmitted via the REST API and sent to the Vue frontend for direct downloading from there. Finally, users will be able to interact with the system using the Vue frontend, which will then connect to the NGINX backend in order to authenticate via the OAuth servers, add participants, or modify studies on the MySQL database.

5. Conclusion

Cardiovascular studies can be very expensive for researchers and inconvenient to participants due to the technology used to collect data for them, meaning that valuable data could be lost. Our project is significant because the use of Fitbit technology could help researchers and the general public alike by allowing studies to collect more data at less of a cost and thus gather more data for researcher use. To help solve this problem we will create a web app that will allow these studies to use Fitbits instead of heart monitors while also mitigating the issues and inefficiencies that existed with the previous technology that our client used.

Of course, to make a web app we needed to figure out what parts will go into it. Some major parts that we needed to address in order to actually create this web app included the database to store the data, the web app's frontend for users to interact with, the web app's backend to run all of the behind-the-scenes code, an API to interact with our database, and our methods of user authentication so users can securely log in and log out. We have conducted research on these different challenges and found what we believe to be the solutions that will best fit our current goals. Based on our findings, we have chosen to use MariaDB as our database, Vue for the frontend, NGINX for the backend, REST for our API, and OAuth2 for our user authentication.

Moving forward, we plan to put all our different solutions together to create a fully functional web app for our client to use. This will include making functioning prototypes using each of the solutions that we have discovered and testing them out in order to see how to best implement them, then moving on to assembling a full prototype with all of the parts together. Finally, we plan to build on that full prototype to finish constructing our web app. We believe that with these steps, we should be able to create a fully functioning web app for our client by our final deadline.